INFORMATION AND CONTROL 67, 126–143 (1985)

On Space and Time Efficient TM Simulations of Some Restricted Classes of PDA's*

OSCAR H. IBARRA

Department of Computer Science, University of Minnesota, Minneapolis, Minnesota 55455

SAM M. KIM

Department of Computer Science, Rensselaer Polytechnic Institute, Troy, New York 12181

AND

LOUIS E. ROSIER

Department of Computer Sciences, University of Texas, Austin, Texas 78712

In this paper we present some space/time efficient Turing machine algorithms for recognizing some subclasses of DCFLs. In particular, we show that the finite minimal stacking and "simple" strict restricted (a subclass of strict restricted) deterministic pushdown automata (FMS-DPDAs, SSR-DPDAs, respectively) can be simulated by offline Turing machines simultaneously in space $S(n)$ and time $n^2/S(n)$ for any tape function $S(n)$ satisfying $\log n \leqslant S(n) \leqslant n$ which is constructable in $n^2/S(n)$ time. Generalizations are then made for the corresponding classes of 2-way DPDAs.

1. INTRODUCTION

The study of context-free languages (CFLs) is an important topic in computer science. Recently, there has been a lot of work finding time and/or space efficient algorithms for recognizing CFLs. It was shown in Lewis, Hartmanis, and Stearns (1965) that an arbitrary CFL can be recognized in $O(\log^2 n)$ space. The algorithm, however, requires $O(n^{\log n})$ time. For the deterministic case an algorithm that runs simultaneously in $O(\log^2 n)$ space and $O(n^2/\log^2 n)$ time is known (Cook, 1979; Verbeek, 1981; von Braumühl *et al.* 1983, 1980). This result generalizes to an algorithm that runs in $S(n)$ space and $n^2/S(n)$ time for any constructable function $S(n)$, satisfying $\log^2 n \leqslant S(n) \leqslant n$. Whether or not the $\log^2 n$ can be reduced is still open.

* This research was supported in part by NSF Grants MCS 81-02853 and MCS 83-04756.

0019-9958/85 $3.00

At present it is not known whether $O(\log n)$ space is sufficient to recognize an arbitrary CFL. This seems unlikely, however, as results in (Richie and Springsteel, 1972; Sudborough, 1975) show that an affirmative answer would imply the equivalence of deterministic and nondeterministic linear bounded automata. It is reasonable to expect, however, that large subclasses of the CFLs are recognizable in $O(\log n)$ space, perhaps even all deterministic CFLs (DCFLs). Many subclasses recognizable in $O(\log n)$ space have been shown recently (Igarashi, 1978, 1977; Lipton and Zalcstein, 1976). Among these are the bracket-languages of Mehlhorn (1976) and the parenthesis languages of Lynch (1977). Also in Igarashi (1978) it was shown that both finite minimal stacking and strict restricted deterministic pushdown automata could be simulated in $O(\log n)$ space. Such machines can recognize deterministic finite turn languages, Dyck languages, standard languages, structured context-free languages, and left-most Szilard languages of phrase structured grammars (see Igarashi, 1978).

In this paper we present some space/time efficient Turing machine algorithms for recognizing some subclasses of DCFLs. In particular, we show that the finite minimal stacking (a generalization of finite-turn, Valiant, 1974) and "simple" strict restricted (a subclass of strict restricted) deterministic pushdown automata (FMS-DPDAs, SSR-DPDAs, respectively) can be simulated by offline Turing machines simultaneously in space $S(n)$ and time $n^2/S(n)$ for any tape function $S(n)$ satisfying $\log n \leqslant S(n) \leqslant n$, which is constructable in $n^2/S(n)$ time. The results are optimal (within a constant factor) for both machine models. To see this we note that the language $L = \{w \# w^r \mid w \in (0+1)^*\}$ is recognizable by both FMS-DPDAs and SSR-DPDAs. If a Turing machine can recognize L in $S(n)$ space and $T(n)$ time, then we can easily construct a 1-tape Turing machine that recognizes L in time $T(n) * S(n)$. Since L requires $O(n^2)$ time for recognition by a 1-tape Turing machine (Hopcroft and Ullman, 1979), we must have $T(n) * S(n) \geqslant n^2$, i.e., $T(n) \geqslant n^2/S(n)$. Furthermore, the recognition of L requires $\log n$ space (*ibid*), so the $\log n$ lower bound for $S(n)$ is also optimal. The $O(\log n)$ space algorithms presented in Igarashi (1978) for finite minimal stacking and strict restricted DPDAs require $O(n^2)$ and $O(n^3)$ time, respectively. For "simple" strict restricted DPDAs the time in (*ibid*), could be reduced to $O(n^2)$ in a straightforward manner. For the case of finite minimal stacking machines we show that one work tape is sufficient when $S(n)$ is between $\log n$ and $n/\log n$. We note that while the SSR-DPDAs are more restricted than the strict restricted DPDAs of (*ibid*), each language shown to be recognizable (in *ibid*) by the strict restricted machines is also recognizable by the "simple" ones. At this time we are unable to generalize this result to the strict restricted case. The construction of SSR-DPDAs for left-most Szilard languages is examined in the Appendix. In both cases the same techniques are then used to give

corresponding results for the linear time bounded 2-way FMS (SSR)DPDAs.

2. Preliminaries

We assume the reader is familiar with the definitions of Turing machines (TMs), and deterministic pushdown automata (DPDAs). Basically, we employ the definitions and notation of DPDAs given in (Igarashi, 1978; Valiant, 1973). The reader should consult these sources, if they are unfamiliar. A DPDA M is a 7-tuple $M = \langle Q, \Sigma, \Gamma, \delta, q_0, Z_0, F \rangle$, where

Q is a (finite) set of states,

Σ is the (finite) input alphabet,

Γ is the (finite) pushdown alphabet,

q_0 is the initial state,

Z_0 (in Γ) is the bottom-of-stack marker

$F \subseteq Q \times \Gamma$ is the set of accepting modes, and

δ is the transition function.

In addition to the usual restrictions placed on δ, in order to insure that the DPDA M has at most one next move defined at each step and is therefore deterministic (see (Ibarra, 1971; Igarashi, 1978; Valiant, 1973)), we assume that δ is such that the DPDA terminates for all inputs.

Now the stack height can grow by at most a constant during any one move of the DPDA (the constant, of course, depends on the machine). For ease of explanation in our subsequent discussion, we assume that this constant is 1. The reader should note, however, that the algorithms given can easily be modified to work for any such constant $\geqslant 1$.

3. A Space and Time Efficient Simulation of FMS-DPDAs

A configuration of the DPDA M is a pair from $Q \times \Gamma^*$. For a configuration C, let $|C|$ denote its stack height. Following Igarashi (1978), let $C \vdash^w C'$ be a derivation, i.e., the sequence of configurations, beginning with C, in which the DPDA reads input w, and ends up in configuration C'. A configuration C_i is said to be a stacking configuration in the derivation $C \vdash^w C'$ if and only if it is not followed by any configuration with stack height less than or equal to $|C_i|$ in the derivation. Let C_0 be the initial configuration. Suppose the machine takes t moves to get from a configuration C_0 to C' while reading w (i.e., $C_0 \vdash^w C'$). Then a stacking configuration C_i

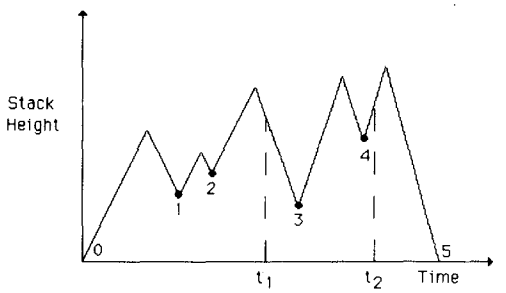

FIG. 1. Minimal stacking configurations.

in the derivation $C_0 -^w C'$ is a minimal stacking configuration in the derivation (from C_0 to C') at time t if and only if one of the following two conditions are met:

(1) C_i is the first stacking configuration in the derivation.

(2) There is a configuration of height $> |C_i|$ between C_i and the stacking configuration immediately preceding C_i in the derivation.

Notice that during the computation C_i may be a minimal stacking configuration at some time t and may or may not be at a later time t'. It is a dynamic property that changes as a computation proceeds. In Fig. 1, for example, points 0, 1, and 2 correspond to minimal stacking configurations at time t_1 while points 0, 3, and 4 represent the minimal stacking configurations at time t_2.

We now define the class of finite minimal stacking DPDAs (FMS-DPDAs).

DEFINITION. A DPDA M (over input alphabet Σ) is a FMS-DPDA if there exists a positive integer k such that for any $w \in \Sigma^*$ the number of finite minimal stacking configurations in the derivation $C_0 -^w C$ is at most k for any such reachable configuration C.

Although the definition of FMS-DPDAs is based on a property of the machine's computation (and not on a static property) it should be noted that it is decidable to determine if an arbitrary DPDA is finite minimal stacking (Igarashi, 1978).

Now we are ready to show the following:

THEOREM 1. *A FMS-DPDA can be simulated by an offline TM with a single worktape in $O(S(n))$ space and $O(n^2/S(n))$ time for any function $S(n)$, where $\log n \leqslant S(n) \leqslant n/\log n$ and $S(n)$ is tape constructable by an offline single tape TM in $n^2/S(n)$ time.*

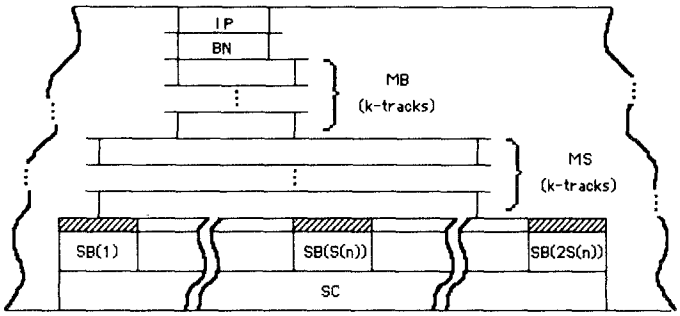

FIG. 2. The worktape of M'.

Proof. Let M be a k-minimal stacking DPDA. Then we will construct an offline single tape TM M' that will simulate M. The idea is to divide the stack of M into $O(n/S(n))$ blocks each of size $S(n)$. At any instant, M' will have at most the two topmost blocks of stack symbols represented on the simulation block (SB) of the worktape. It will be used as a "mini-stack" during the simulation. Along with the simulation block SB, the worktape is organized into multitracks which will contain other information such as the input head position, the block number and the current minimal stacking information which is required for the simulation.

We let a stacking configuration of M be denoted by a 5-tuple (A, Q, I, B, S) where A is a stack symbol, Q a state, I the input head position, B the block number, and S is the offset of the position of the stack symbol into the block. Now we are ready to present the organization of the worktape for M' in detail as shown in Fig. 2. We use the following notation:

IP: current input position

BN: current block number

MB: pairs of input head positions and the states each corresponding to a minimal stacking point within the current blocks. Each pair is stored on a separate track. There are k tracks for this. These tracks will behave like a pushdown stack.

MS: Like MB, MS has k tracks. Each track is capable of containing a minimal stacking configuration (A, Q, I, B, S) not associated with the current block. These tracks also function as a LIFO structure.

SB: This track is used for the stack blocks. The cells in SB can be thought of as being indexed 1 through $2 * S(n)$. There are boundary markers at SB(1), SB($S(n)$), and SB($2 * S(n)$). There is a subtrack for the markers to indicate minimal stacking points within the active blocks.

SC: This track is used for scratch (i.e., work) space.

M' will simulate a move of M using its input head to read the input and the SB as the stack (or more precisely as a window into the stack). After each move of the simulation, if the stack height of M changes (thus the SB position representing the top of the stack changes), then M' will move all the information requiring $O(\log n)$ bits on the other tracks, a position in the same direction, and then update the input head position, IP. This insures that this information is always "close" to the worktape head, and hence updating the counter on each step of the simulation does not take too long ($O(\log n)$ time to be exact). The fact that M is k-minimal stacking will allow M' to regenerate the other blocks, when they are needed, using at most $O(\log n)$ additional space. The operation of M' will be divided into $O(n/S(n))$ phases. At the beginning of a phase the top block of $S(n)$ stack symbols will be represented on the lower half of SB and the remaining upper half of it is used for growth of the stack. Thus M' can simulate at least $S(n)$ moves before the simulation requires a stack symbol from another block of the stack. A phase ends when the next required stack position is not available on the blocks currently on the SB. At this time some informational bookkeeping and block restoration must follow before the next phase of the simulation takes place. During each phase the *new* minimal stacking configurations (i.e., the ones which occur during the phase), if any, are recorded by keeping their states and input head positions in the MB, and by marking the corresponding minimal stacking points on the (SB) block. Whenever a marked position is popped the MB is also popped, since the position no longer represents a FMS configuration. Thus, at all times the number of items stored in the MB is equal to the number of marked positions in the SB. If a phase ends with the SB full, then the minimal stacking information from the SB blocks (and MB) is moved to the dedicated tracks (MS) which keeps the current minimal stacking information, to be used for the block reconstruction. (This can easily be accomphished given the information in MB, the marked positions of the SB and the current value of BN.) Then the SB is erased. If a phase ends with the SB empty, there is no additional information to be saved. In either case, the block contents for the next phase of the simulation is then restored on the lower half of SB using the minimal stacking information now contained in the MS.

Figure 3 illustrates how the next block (block 2) is reconstructed, when the phase with block 3 ends at time t with the SB empty. M' searches the current minimal stacking information (contained in the MS, which at this time represents points 0, 1, 2, 4, 6, and 7 of Fig. 3a), and writes the stack symbol from each of the minimal stacking points which occurs in block 2 (points 4, 6, and 7) on their corresponding positions in the SB and marks them (see Fig. 3c). Starting with the minimal stacking point of the height of the current block, if any, or with the next lower one otherwise (2 in this

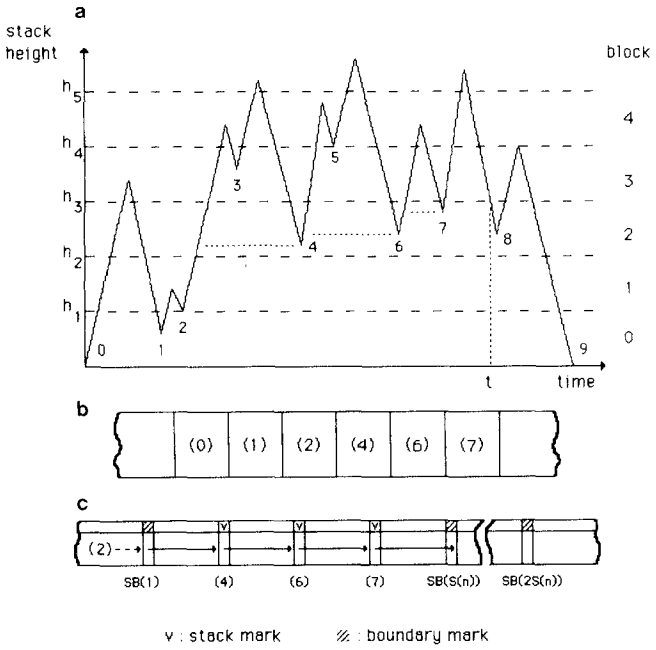

FIG. 3. Block restoration: (a) time–space profile; (b) minimal stacking information at time t (contained in MG); (c) the restoration for block 2.

case), M is simulated until the stack height reaches the current block (h_2). Then M', using SB as the stack, continues the simulation until it meets the next minimal stacking point (point 4 of Fig. 3c), where upon it will use the information corresponding to that minimal stacking point, to resume the simulation from that time. This process is repeated until M' finally meets the block boundary, (the center mark of SB). Notice that the sequence of moves made from one of the minimal stacking points to another consists only of pushing or rewriting moves. Now that the complete contents of the topmost block are available on the SB we are ready for the next simulation phase.

Now we present the algorithm for the simulation:

begin

(//MS, MB, and BN are globals. Assume that, when a step of M is simulated, the work head is on SB top. q is the current state of M. Push $S(A)$ means that A is pushed on top of a stack S.//)

(1) IP := 0; BN := 0; $q := q_0$; SB(1) := Z_0;

(2) push MS(Z_0, q_0, 0, 0, 0) (//This is the first minimal stacking configuration.//)

```
repeat
  (3) simulate M with q, an input symbol at IP, and SB top;
      case
  (4) M pops:
        update q and IP;
        if SB underflows (i.e., moves off the left boundary) then decrement
          BN; call BLOCK-RESTORE;
        else if a stack marker on SB is erased then
            pop MB;
            endif
        endif;
  (5) M pushes:
        if a new minimal stacking point is generated then push MB(IP, q)
          and write a stack marker on SB top;
        endif;
        updare q and IP;
        if SB overflows (i.e., moves off the right boundary) then take the
          stacking configurations from SB and MB, and push them on
          MS; increment BN; call BLOCK-RESTORE;
        endif;
  (6) M rewrites:
        update q and IP;
  endcase;
until (M halts);
end (//algorithm//)

Procedure BLOCK-RESTORE;
(//MS, MB, BN, and SB are globals//)
begin
  (1) Pop all the stacking configurations (i.e., tuples of (A, Q, I, B, S))
      from MS which belong to the current block BN, restore (I, Q) pairs
      on MB and A's to the minimal stacking points on SB (i.e., A is written
      on SB(S)); The minimal stacking points on SB are also marked at this
      time;
  (2) With the stacking configuration from MS top, simulate M until the
      stack height reaches the current block BN (Note that all such
      simulated moves are stacking moves);
  (3) With the current configuration from step (2) above and the minimal
      stacking configurations restored in step (1) above, simulate M and
      restore the contents of block BN on SB(1) through SB(S(n)) (as
      described in the example concerning Fig. 3);
end (//BLOCK-RESTORE//)
```

For the execution time of the main program, steps (1) and (2) run in

constant time. The global time needed for step (3) and step (6) is no more than $O(n \log n)$. It is easily seen that the total time needed for updating the MS in step (5) above is $O(S(n) \log S(n))$. The time analysis of BLOCK-RESTORE can be observed from the following:

(i) Step (1) (the marking of the block) needs $O(S(n) \log S(n))$ time,

(ii) Step (2) needs $O(n)$ time, and

(iii) Step (3) (the actual restoration for the block) needs $O(S(n))$ time.

Since those subroutines are called at most $O(n/S(n))$ times, steps (4) and (5) of the main routine need at most $O(n/S(n) * (n + S(n) \log S(n))) = O(n^2/S(n) + n \log S(n))$ steps. So the overall time is $O(n^2/S(n) + n \log n)$, i.e., $O(n^2/S(n))$ if $S(n) \leqslant n/\log n$ and $O(n \log n)$ otherwise. ▮

It seems difficult to achieve the same time bound $O(n^2/S(n))$ when the space is in the range $n/(\log n) < S(n) \leqslant n$ with only one worktape. The bottleneck seems to be the global time of $O(n \log S(n))$ needed to count the displacement on the block and $O(n \log n)$ to update the stacking points MB and the input position counter IP. With a multitape TM, however, we can achieve the time $O(n^2/S(n))$ for the whole range. For the upper range we can simply use the algorithm for general DPDAs by von Braumühl and Verbeek (1980); see also von Braumühl *et al.*, 1983).

But here we sketch a simpler algorithm for FMS-DPDAs, that achieves the goal for the entire range. The multitape TM has dedicated tapes for each information block IP, BN, MB, MS, and SB (see Fig. 2). It will also have an additional $k + 2$ tapes including a scratch tape. Instead of updating MB for each new minimal stacking point, the TM does it after each time slot of length $S(n)$. The TM uses k tapes to record the input head displacement for each minimal stacking point generated on the SB (during a phase). It uses an extra tape to count $S(n)$ steps. All the counting on these tapes is done in unary. Along with the stacking mark on the SB, the state corresponding to each minimal stacking point is also kept.

Let K_i, $1 \leqslant i \leqslant k$ be the tape to record the input displacement for the ith minimal stacking point within the current phase of the simulation, and K_T be the counter tape for the time slot. The TM works basically the same way as the single tape TM except for the following operations: While the head on SB simulates M, the tapes K_T and K_i, $1 \leqslant i \leqslant k$ count the processing steps and the input head displacement respectively (in unary by their displacements from the left boundary). When the ith minimal stacking point is generated on the mini-stack SB, K_i marks the current position. Later, if the ith minimal stacking point on SB is erased, the head K_i goes back to the mark, erases it, and returns to the current point to resume the counting. If the tape K_T meets the right endmarker (i.e., the phase of $S(n)$

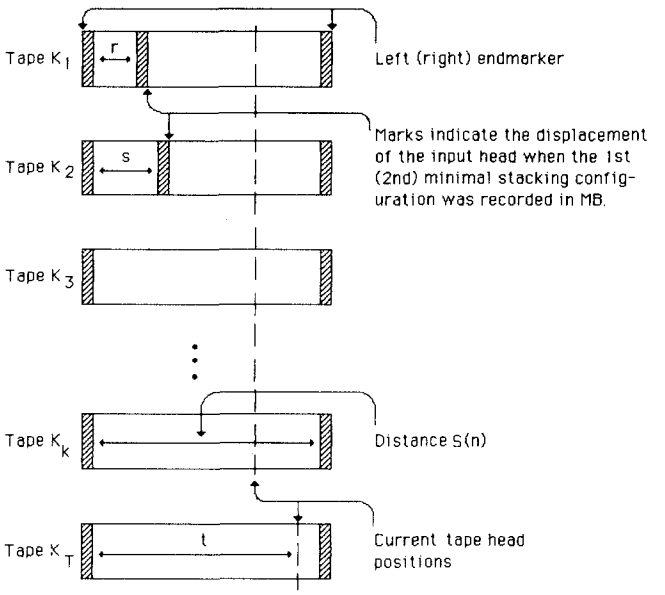

FIG. 4. Tapes used to record the input head displacements for each possible minimal stacking configuration generated during a phase.

moves is over) the TM will then update MB with the state and input position corresponding to each stack mark on SB. The input position for the ith minimal stacking point is simply computed from the contents of the IP and the displacement of the marker from endmarker of the tape K_i. IP is updated with the contents of K_i each time a $S(n)$ length simulation phase is over. Notice that, except for the markers, the contents of all the K_i tapes are the same at any point of the computation. So we may use any counter K_i, $1 \leqslant i \leqslant k$, to update IP. (For example, consider the configuration of tapes K_T, K_i $(1 \leqslant i \leqslant k)$ shown in Fig. 4. At the time represented, exactly t steps have occurred in a phase. In addition, there are 2 minimal stacking configurations recorded in the MB. The respective input head displacements are then r and s.) When the SB becomes empty or overflowed, the procedures for updating the minimal stacking information (MS), MB and reconstructing the next block on the SB are basically the same as those for the single tape case and the details can easily be supplied by the reader.

Now the time required to update MB and IP will be no more than $O(S(n))$ for each time slot. Counting the displacement in a block will also be done in $O(S(n))$ time. So the global time for these operations is $O(n/S(n) * S(n)) = O(n)$. The other cost to be considered is the time for the tapes K_1 through K_k to manipulate the stacking point markers. Let t_i be the global time the tape K_i takes for the manipulation. The reader can

show that the time used during any time slot for the manipulation of K_i is $O(S(n))$. It follows therefore that

$$\sum_{i=1}^{k} t_i \leqslant O(n).$$

Since the remaining dominating factor is $O(n)$ steps for block restoration which occurs for each phase, the overall time is $O(n/S(n) * n) = O(n^2/S(n))$.

Now one can see that there was nothing special in the previous proof that required the DPDAs input head to be 1-way except for the fact that the run time was implicitly assumed to be $O(n)$. Consequently, we also have the following generalized result for linear time bounded 2-way FMS-DPDAs. (2-way DPDAs are essentially DPDAs whose input head is allowed to move in both directions. See Cook, 1971; Galil, 1977, for a formal definition.) Such machines can accept noncontext-free languages such as $\{xx \mid x \in \Sigma^*\}$ and $\{a^{2^n} \mid n \geqslant 0\}$. (The first language can be recognized by a 2-way finite-turn DPDA but the second seems to require the power of a 2-way FMS-DPDA.) Note that in contrast to the case of 1-way FMS-DPDAs, it is easily shown that it is undecidable to determine whether an arbitrary 2-way DPDA is finite minimal stacking. (It is also undecidable to determine if an arbitrary 2-way DPDA is finite-turn.)

THEOREM 2. *A linear time bounded 2-way FMS-DPDA can be simulated by an offline TM with a single worktape in $O(S(n))$ space and $O(n^2/S(n))$ time for any function $S(n)$, where $\log n \leqslant S(n) \leqslant n/\log n$ and $S(n)$ is tape constructable by an offline single tape TM in $n^2/S(n)$ time.*

The above theorem gives rise to the question of whether linear time bounded 2-way FMS-DPDAs are less powerful, in terms of recognizing ability, than nonlinear such machines. We feel that this is the case but are unable to provide a proof at this time. However, we present the following conjecture. Consider the language $L = \{xx^r y \mid x, y \in \Sigma^+\}$. A 2-way DPDA to accept this language was described in Cook (1971) as follows: "Very briefly the 2-way DPDA copies the input string onto the pushdown store and moves the input head back to the left. Then the input string (left to right) is compared with the pushdown store symbol by symbol until a discrepancy is found. The pushdown store is then restored by moving the input head back left and copying. The pushdown store then pops one, and the process is repeated. If the pushdown store is ever emptied while the input head is away from the left end, the input is accepted." Note, however, that the machine constructed in Cook (1971) is not finite minimal stacking and that its run time is quadratic. Let this machine be M. Now M can be altered as follows to construct M_k, a 2-way k-minimal stacking DPDA.

The finite state control of M_k, in addition to remembering the state of M, also can store a number between 0 and k. The number stored in the finite state control of M_k at the beginning of a computation is k. Now on each input M_k simulates the computation of M. Whenever M in the computation enters a new stacking configuration (it does this whenever the previous more was a pop and the current move increases the size of the pushdown store), M_k marks the current position on the stack with a special symbol and decrements the number stored by 1 (providing it is currently nonzero—if it is currently 0, M_k rejects the input). The marked position, at this time of the computation, is a minimal stacking point. Whenever M_k detects that the topmost stack symbol is the special symbol it pops the symbol and increments the number by 1. (The marked position no longer corresponds to a minimal stacking point.) M_k accepts an input whenever M, in the simulation does. Thus, for each k, $L(M_k) \subseteq L(M)$ and M_k is a k-minimal stacking DPDA. M_k does not, however, run in linear time. (M_k needs quadratic time on all input strings of the form $0^n1^20^n0^n$, for example.) We then conjecture that for some k, $L(M_k)$ is not accepted by a linear time bounded 2-way FMS-DPDA.

4. Simple Strict Restricted DPDAs

In this section, we introduce the simple strict restricted DPDAs (SSR-DPDAs) and show that they can be simulated by an offline single tape TM simultaneously in $S(n)$ space and $O(n^2/S(n))$ time for any "nice" function, $\log(n) \leqslant S(n) \leqslant n$.

Definition. A DPDA $M = \langle Q, \Sigma, \Gamma, \delta, q_0, Z_0, F \rangle$ is called simple strict restricted if both of the following conditions are met:

(1) If $\delta(q, \varepsilon, A)$ is defined, then for all $B \in \Gamma$ and $a \in \Sigma$, $\delta(q, a, B)$ is not defined.

(2) If $\delta(q, \pi, A) = (q', \alpha)$ and $\delta(q, \pi, B) = (q'', \gamma)$, then either

(i) $\alpha^{(1)} = \gamma^{(1)}$ and $|\alpha| = |\gamma|$ and $q' = q''$ or

(ii) q' or $q'' \in D$

where A and B are arbitrary elements of Γ, q an arbitrary state, $\pi \in \Sigma \cup \{\varepsilon\}$, D is a set of dead states from which nothing may be accepted, and $\alpha^{(1)}$ is the topmost symbol of the string α in the stack. The reader should note the difference between this definition and that of the strict restricted DPDA defined in Igarashi (1978). The restriction, in 2(i) above, requiring $\alpha^{(1)}$ to be the same as $\gamma^{(1)}$ is the only difference between the SSR-DPDA and the SR-DPDA of *ibid.* The languages presented in *ibid.* as

examples of languages accepted by SR-DPDAs (i.e., Dyck languages, Standard languages, structured context-free languages (Igarashi, 1978; Richie and Sprinsteel, 1972) and leftmost Szilard languages of phrase structured grammers (Igarashi, 1977), are also accepted by SSR-DPDAs. The Appendix shows the construction of SSR-DPDAs for leftmost Szilard languages. We can easily see that the SR-DPDAs given in (Igarashi, 1978) for structured context-free languages are SSR-DPDAs. Constructing SSR-DPDAs for Dyck languages and Standard languages is straightforward.

We are now ready to show the following:

THEOREM 3. *A SSR-DPDA M can be simulated by an offline single tape TM M' in $O(S(n))$ space and $O(n^2/S(n))$ time for any function $S(n)$ where $\log n \leqslant S(n) \leqslant n$ whenever $S(n)$ is tape constructible by an offline single tape TM in $O(n^2/S(n))$ time.*

Proof. Again, as in Theorem 1, we divide the stack into $O(n/S(n))$ blocks each of size $S(n)$. The simulation used in Theorem 1 was such that the moves of a given block were those moves which directly followed the moves of the previous phase in the computation. Thus any given block of the stack may be present in the mini-stack during different phases of the simulation. However in this simulation, we simulate all the moves which occur within a given block of the stack in a single phase. Thus, if we let Fig. 5 represent a computation, then all moves which M takes with the stack top in block i (i.e., those corresponding to the dark line segments in the figure) are simulated in the same phase. To do this, we assume that if a dead state is entered then it is not entered until the stack top is in the block currently being processed. The simulation therefore involves looking for the move where M enters a dead state.

Let a partial configuration (P-C, for short) of M be a 4-tuple (q, w, i, h),

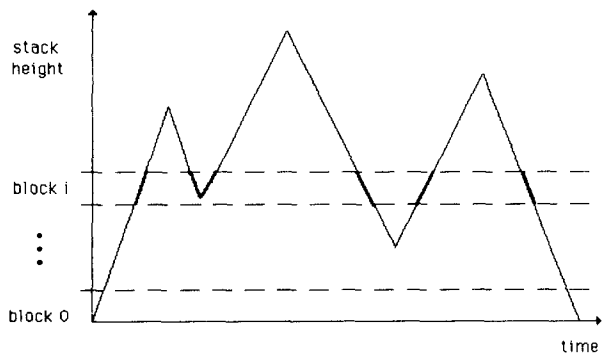

FIG. 5. A computation of M.

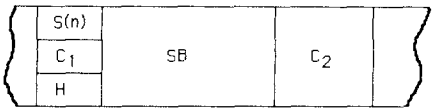

FIG. 6. The worktape of M'.

where q is the state of M, w an input string, i the input head position and h the stack height. From the two restrictions the SSR-DPDA M has, it can be seen that M', given a valid P-C from a computation of M, can trace the exact sequence of the P-Cs up to any possible partial configuration, in $O(\log n)$ space and $O(n)$ time. Furthermore, M' can compute the exact symbol M will write in that configuration at any height.

For the simulation on block i, M' starts with the initial P-C $(q_0, w, 0, 0)$ and traces the P-C until block i is entered. Then M' will initiate the actual simulation on the block using a worktape as a mini-stack. If M' crosses the left or the right boundary of the block, it quits the actual simulation phase and traces the partial configuration until M's stack enters the block again. A block simulation phase will be completed when there is no more input. Then the simulation for the next higher block is initiated. The simulation continues until M rejects or M' sees a block simulation phase where no configuration occurs within that block.

For an efficient simulation, the worktape of M' is organized as in Fig. 6, where SB is the mini-stack of size $S(n)$, H the current block height, which is the stack height of SB(1), and C_1 and C_2 are counters for downward displacement and upward displacement from SB, respectively. From C_1, C_2, and H, we can assertain the top position of the current stack.

Now we present the algorithm for the simulation:

```
begin
(//i, q and π are block number, current state and current input symbol
(possibly ε), respectively. Initially SB has blanks//)
  SB(1) := Z0; H := 1;
  done := true; accept := false; reject := false;

  repeat
    C1 := H-1; C2 := 0; q := q0;
    bring the input head to the first input symbol;

  while q is not a halting state do
    if (C1 = 0 and C2 = 0) then (//stack top is in current block//)
      begin
        with q, π, and SB, simulate a move of M;
        if SB underflows then update C1
          else if SB overflows then update C2 endif
```

```
      endif;
      done := false;
    end
  else (//stack top is not in current block//)
    begin
      simulate M with q, π and a stack symbol B which does not make
      the next state dead; (//if there is no such B, then the next state q
      becomes a dead state.//)
      if C1 ≠ 0 then
            begin
              update C1;
              if C1 = 0 then (//stack top is back on SB//)
                update SB;
              endif
            end
          else update C2
        endif;
      end
    endif
  endwhile;

    if q is a dead state then
      reject := true
    else
      if done then accept := true
        else done := true; H := H + S(n); (//ready for the next phase//)
      endif
    endif
  until (reject or accept) end.
```

Using techniques presented in Fischer, Meyer, and Rosenberg (1968), the manipulation of counters C_1 and C_2 can be accomplished in $O(n)$ time for each phase. Since there are $O(n/S(n))$ phases, the overall time is $O(n^2/S(n))$. ∎

Once again, there was nothing special in the previous proof which required the DPDAs input head to be 1-way except that this implicitly meant that the run time was linear. Consequently, we also have the following generalized result for linear time bounded 2-way SSR-DPDAs.

THEOREM 4. *A linear time bounded 2-way SSR-DPDA can be simulated by an offline single tape TM M′ in* $O(S(n))$ *space and* $O(n^2/S(n))$ *time for any function* $S(n)$ *where* $\log n \leqslant S(n) \leqslant n$ *whenever* $S(n)$ *is tape constructible by an offline single tape TM in* $O(n^2/S(n))$ *time.*

APPENDIX: CONSTRUCTION OF SSR-DPDAs FOR LEFTMOST SZILARD LANGUAGES

Let $G=(V_N, V_T, P, S)$ be a phase-structured grammar, and $V=V_N \cup V_T$. Let $xuy \to_\pi xvy$ denote a derivation according to a production rule $u \to v$ named π, where $x, y, v \in V^*$, $u \in V^* V_N V^*$. A leftmost derivation is a derivation which has restrictions such that $x \in V_T^*$ and $u \in V_N^+$. Suppose that $w_0, w_1, \ldots, w_{r-1}$ are in $V^* V_N V^*$, and w_r is in V^*, for $r \geqslant 1$. If there is an $\alpha = \pi_1 \pi_2 \cdots \pi_r$ such that $w_0 \to_{\pi_1} w_1$, $w_1 \to_{\pi_2} w_2, \ldots$, and $w_{r-1} \to_{\pi_r} w_r$, then we say $w_0 \to_\alpha^* w_r$ or w_r is derived from w_0 with α. We call α the associate word of the derivation (Igarashi, 1977).

Formally a Szilard language $\mathrm{SZ}(G)$ of a grammar $G(V_N, V_T, P, S)$ is defined as $\mathrm{SZ}(G)=\{\alpha \mid S \to_\alpha^* w,\ w \text{ in } V_T^*,\ \alpha \text{ in } P^*\}$. If all derivations are restricted to be leftmost, then the language is called a leftmost Szilard language, which we denote as $\mathrm{SZ_L}(G)$.

Moriya (1973) proved that $\mathrm{SZ_L}(G)$ of a phrase-structured grammar G is a CFL by constructing the associated PDA. But his PDA is not a SSR-DPDA. We, therefore, construct a SSR-DPDA M which recognizes $\mathrm{SZ_L}(G)$. Let $w=\pi_1 \pi_2 \cdots \pi_n$ be an input. Suppose π_{i-1} is $\alpha_1 \to \beta_1$ and π_i is $\alpha_2 \to \beta_2$, $1 < i \leqslant n$. The idea is to store β_1 in the stack and α_2 in a buffer, and check if there is any topmost nonterminal substring of the stack which is equal to α_2 in the buffer. If they match, then, with β_2 pushed on the stack and the left side of the production rule π_{i+1} stored in the buffer, the process is repeated.

Given a grammer $G=(V_N, V_T, P, S)$, a SSR-DPDA M is formally constructed as follows:

$$M=\langle Q, \Sigma, \Gamma, \delta, q_0, Z_0, F\rangle,$$

where $Q=Q_1 \cup \cdots \cup Q_k \cup \{q_d, q_f\}$, $k=|P|$, $Q_i=\{q_{i_\alpha} \mid \pi_i = u \to v,\ \pi_i \in P,$ and α is a suffix of $u\}$, $\Sigma=\{\pi_i \mid \pi_i \in P\}$, $\Gamma=V_N \cup \{\bar{A} \mid A \in V_N\} \cup \{Z_0\}$, $F=\{q_f\}$.

Let $\|\beta\|$ denote the operation of deleting terminal symbols from a sentential form β after marking the nonterminal symbols which have a terminal symbol next to the right. For example, suppose $V_T=\{a, b, c, d\}$ and $V_N=\{A, B, C\}$, then $\|abacABcabBCaA\| = A\bar{B}B\bar{C}A$, i.e., a bar indicates that a terminal symbol is deleted from the right neighboring position.

Now we define δ as follows:

(1) $\delta(q_0, \pi, Z_0)=(q, \|v\| Z_0)$ if $\pi = S \to v$.

(2) $\delta(q, \pi_i, Z)=(q_{i_u}, Z)$ and $\delta(q, \pi_i, \bar{Z})=(q_{i_u}, \bar{Z})$ if $\pi_i = u \to v$.

(3) $\delta(q_{i_u}, \varepsilon, Z)=(q_{i_{u'}}, \varepsilon)$ if $u=Zu'$, $u' \in V_N^*$. Also, $q_{i_{u'}}=q_i$, if $u'=\varepsilon$. $\delta(q_{i_u}, \varepsilon, \bar{Z})=(q_i, \varepsilon)$ if $u=Z$.

(4) $\delta(q_i, \varepsilon, Z) = (q, \|v\| Z)$ and $\delta(q_i, \varepsilon, \bar{Z}) = (q, \|v\| \bar{Z})$ if $\pi_i = u \to v$.

(5) $\delta(q, \varepsilon, Z_0) = (q_f, Z_0)$.

(6) For the other cases which are not defined above, the machine enters state q_d which is a dead state.

From the above construction, it can be seen that M is a SSR-DPDA as described in Section 4.

ACKNOWLEDGMENT

We would like to thank the referees for suggestions which improved the presentation of our results.

RECEIVED November 14, 1984; ACCEPTED July 25, 1985

REFERENCES

COOK, S. (1979), Deterministic CFL's are accepted simultaneously in polynomial time and log squared space," *in* "Proc. 11th ACM Sympos. on Theory of Comput.," pp. 338–345.

COOK, S. (1971), Linear time simulation of deterministic two-way pushdown automata, *in* "Proc. 1971 Internat. Fed. Inform. Process Congress," pp. 75–80, North-Holland, Amsterdam.

FISCHER, P., MEYER, A., AND ROSENBERG, A. (1968), Counter machines and counter languages, *Math. Systems Theory* **2**, No. 3, 265–283.

GALIL, Z. (1977), Two-way deterministic pushdown automaton languages and some open problems in the theory of computing, *Math. Systems Theory* **10**, 211–228.

HOPCROFT, J. AND ULLMAN, J. (1979), "Introduction to Automata Theory, Languages, and Computation, *Addison–Wesley*, Reading, Mass.

IBARRA, O. (1971), Characterizations of some tape and time complexity classes of Turing Machines in terms of multihead and auxiliary stack automata, *J. Comput. System Sci.* **5**, No. 2, 88–117.

IGARASHI, Y. (1978), Tape bounds for some subclasses of deterministic context-free languages, *Inform. Contr.* **37**, 321–333.

IGARASHI, Y. (1977), The tape complexity of some classes of Szilard languages, *SIAM J. Comput.* **6**, No. 3, 461–466.

LEWIS, P., HARTMANIS, J., AND STEARNS, R. (1965), Memory bounds for the recognition of context-free and context-sensitive languages, *in* "IEEE Conf. Record on Switching Circuit Theory and Logic Design," pp. 191–202.

LIPTON, R., AND ZALCSTEIN, Y. (1976), "Word Problems Solvable in Logspace, "Computer Science Department Tech. Report 6, Yale University.

LYNCH, N. (1977), Logspace recognition and translation of parenthesis languages, *J. Assoc. Comput. Mach.* **24**, No. 4, 583–590.

MEHLHORN, K. (1976), Bracket-languages are recognizable in logarithmic space, *Inform. Process. Lett.* **5**, No. 6, 168–170.

MORIYA, E. (1973), Associate languages and derivational complexity of formal grammars and languages, *Inform. Contr.* **22**, 139–162.

RICHIE, R., AND SPRINGSTEEL, F., (1972), Language recognition by marking automata, *Inform. Contr.* **20**, 313–330.

SUDBOROUGH, I. (1975), A note on tape-bounded complexity classes and linear context-free languages, *J. Assoc. Comput. Mach.* **22**, No. 4, 499–500.

VALIANT, L. (1973), "Decision Problems for Families of Deterministic Pushdown Automata," Ph. D. thesis, University of Warwick, U. K.

VALIANT, L. (1974), The equivalence problem for deterministic finite-turn pushdown automata, *Inform. Contr.* **25**, 123–133.

VERBEEK, R. (1981), Time–space trade-offs for general recursion, *in* "Proc. 22nd IEEE Found. of Comput. Sci.," 228–234.

VON BRAUMÜHL, B., COOK, S., MEHLHORN, K., AND VERBEEK, R. (1983), The recognition of deterministic CFL's in small time and space, *Inform. Contr.* **56**, 34–51.

VON BRAUMÜHL, B., AND VERBEEK, R. (1980), A recognition algorithm for deterministic CFLs optimal in time and space, *in* "Proc. 21st IEEE Found of Comput. Sci.," pp. 411–420.

643/67/1-3-10